Elimination des symétries locales durant la résolution dans les CSPs

Belaïd Benhamou, Mohamed Reda Saïdi

Elimination des symétries locales durant la résolution dans les CSPs

Belaïd Benhamou **Mohamed Réda Saïdi**

Laboratoire des Sciences de l'Information et des Systèmes (LSIS)
Centre de Mathématiques et d'Informatique
39, rue Joliot Curie - 13453 Marseille cedex 13, France
{Belaid.Benhamou, saidi}@cmi.univ-mrs.fr

Résumé

Plusieurs approches exploitant l'élimination des symétries dans la résolution des CSPs sont apparues récemment. La grande majorité de ces méthodes exploitent les symétries globales [1] du problème étudié et ne tente pas d'exploiter les symétries locales[2]. Il a été montré que l'élimination des symétries globales peut être utile dans la résolution des CSPs. Mais exploiter uniquement ces symétries peut ne pas suffire pour résoudre des problèmes difficiles contenant de nombreuses symétries locales. En effet, un problème peut avoir peu ou pas du tout de symétries initiales (globales) et devenir très symétrique à certains noeuds durant la recherche. Dans ce papier, nous étudions le principe général de la symétrie sémantique et on définit la symétrie syntaxique qui est une condition suffisante de la symétrie sémantique. Nous montrons comment la symétrie syntaxique est détectée et éliminée localement pour améliorer l'efficacité des méthodes de résolution de CSPs. Les expérimentations confirment que l'exploitation des symétries locales est profitable dans la résolution des CSPs.

Abstract

Many research works on symmetry in CSPs appeared recently. But, most of them deal only with the global symmetry[3] of the studied problem and give no strategy that can be used to detect and eliminate local symmetry[4]. Eliminating global symmetry is shown to be useful, but exploiting only these symmetries could be not sufficient to solve some hard locally symmetrical problems. That is, a problem can have few or no initial symmetries and become very symmetrical at some nodes during the search. In this paper we study a general principle of semantic symmetry and define a syntactic symmetry form which is a sufficient condition for semantic symmetry. We show how this syntactic symmetry is detected and eliminated locally to increase CSP tree search methods efficiency. Experiments confirm that local symmetry breaking is profitable for CSP solving.

[1] les symétries initiales du problème apparaissant à la racine de l'arbre de recherche

[2] les symétries du CSP résultant à un noeud de l'arbre de recherche correspondant à une instanciation partielle

[3] The symmetry of the initial problem appearing at the root of the search tree

[4] The symmetry of the resulting CSP at a node of the search tree corresponding to a partial instantiation

1 Introduction

A notre connaissance le principe de symétrie a été introduit pour la première fois par Krishnamurty [20] pour optimiser la résolution dans la logique propositionnelle. La symétrie pour les contraintes booléennes a été étudiée en profondeur dans [5, 6, 7], les auteurs ont montré comment les détecter et ont prouvé que leur exploitation est une réelle amélioration de l'efficacité de plusieurs algorithmes de déduction automatique. La notion d'interchangeabilité dans les CSPs a été introduite dans [14] et la symétrie pour les CSPs a été étudiée dans [27, 4].

Et de puis, plusieurs recherches sur la symétrie sont apparues. Par exemple, l'approche statique utilisée par James Crawford et al. dans [10] pour la théorie de logique propositionnelle consiste à ajouter des contraintes pour casser les symétries globales du problème. Cette technique a été améliorée dans [1] et étendue à la programmation logique entière en 0-1 dans [2]. Une technique similaire a été utilisée par Masayuki Fujita et al. dans [21] pour la recherche de modèles dans les problèmes quasi-groupes.

Comme un grand nombre de contraintes peuvent être ajoutées, certains auteurs ont proposé d'ajouter des

contraintes durant la recherche. Dans [3, 17, 18], les auteurs proposent de poster des contraintes conditionnelles qui enlèvent le symétrique d'une interprétation partielle au moment du backtrack. Dans [13, 12, 29, 15], les auteurs suggèrent d'utiliser chaque sous-arbre comme no-good afin d'éviter l' exploration de certaines interprétations symétriques et le concept d'arbre de groupe d'équivalence pour l'élimination des valeurs symétriques est introduit dans [33]. Ces techniques sont appelées respectivement, SBDS, SBDD et GE-Tree.

Récemment une méthode qui élimine les symétries entres variables liées par une contrainte Alldiff est étudiée dans [31], une méthode qui élimine toutes les valeurs symétriques dans les problèmes de surjection est donnée dans [30]. Une synthèse de toutes les définitions connues sur *la symétrie de solution* ou sur *la symétrie de contrainte* est donnée dans [9].

Plus récemment, dans [32], contrairement à la approche statique du lex contrainte de [10], Puget utilise un ordre dynamique sur les variables tout en n'excluant pas la première solution qui pourrait être trouvée, et dans [35] Walsh étudie plusieurs nouveaux propagateurs pour éliminer différentes symétries, parmi eux un qui permet d'éliminer les symétries opérant simultanément sur des valeurs et des variables. En enlevant de façon statique toutes les interprétations symétriques, Gilles Dequen et Olivier Dubois ont prouvé la non existence du quasi-groupe QG2 d'ordre 10 [11]. Finalement, dans le même esprit, Pedro Meseguer et Carme Torras ont proposé une heuristique pour éliminer les symétries durant la recherche [25]. Cependant toutes ces approches exploitent uniquement les symétries globales du problème initial et aucune méthode permettant d'éliminer les symétries locales n'est proposée. Récemment, certains chercheurs ont appelé ces symétries, les symétries conditionnelles [16, 36].

Dans ce papier nous développons le concept général de *symétrie sémantique* pour les CSPs que nous avons initié dans [4] et dont on a présenté récemment une version simplifiée à la conférence sur la symétrie (ISC'07) [8]. Nous avons aussi étudié et étendu le principe de la *symétrie syntaxique* que l'on prouve être une condition suffisante pour la *symétrie sémantique*. Nous montrons comment la symétrie locale est détectée et éliminée durant la recherche, et nous montrons comment l'élimination de la symétrie permet de réduire l'espace de recherche des méthodes de résolution.

Ce papier est organisé comme suite : les rappels sur le CSP et les permutations sont donnés dans la Section 2. La notion de *symétrie sémantique* est définie dans la Section 3. Dans la Section 4 nous discutons la notion de *symétrie syntaxique* qui est une condition suffisante de *symétrie sémantique*. On montre dans la Section 5 comment la symétrie est détectée et éliminée localement durant la recherche et comment une méthode comme Forward Checking exploite ceci pour réduire l'espace de recherche. Dans la Section 6 nous évaluons les techniques proposées par des Expérimentations et la Section 7 conclus ce travail.

2 Préliminaires

2.1 CSPs

Un CSP est un quadruplet $P = (V, D, C, R)$ où : $V = \{v_1, ..., v_n\}$ est un ensemble de n variables ; $D = \{D_1, ..., D_n\}$ est l'ensemble des domaines associés aux variables, D_i inclut l'ensemble des valeurs possibles pour la variable v_i ; $C = \{C_1, ..., C_m\}$ est l'ensemble de m contraintes reliant deux ou plusieurs variables. Une contrainte binaire est une contrainte qui relie exactement deux variables ; $R = \{r_1, ..., r_m\}$ est l'ensemble des relations correspondant aux contraintes de C, r_i représente la liste des tuples permis par la contrainte C_i. Un CSP binaire $\mathcal{P}$ (CSP ne faisant intervenir que des contraintes binaires) peut être représenté par un graphe de contraintes $G(V, E)$ où l'ensemble des sommets V est l'ensemble des variables du CSP et chaque arête $(v_i, v_j) \in E$ connecte les variables v_i et v_j reliées par la même contrainte $C_i \in C$.

La microstructure [14, 19] d'un CSP binaire $\mathcal{P}$ est un graphe $\mathcal{M}_{\mathcal{P}}(V \times D, \acute{E})$, où un sommet de ce graphe représente un couple (variable,valeur) et où chaque arête de $\acute{E}$ correspond soit à un tuple permis par une des contraintes du CSP, soit par un tuple permis car il n'y pas de contrainte entre les variables correspondantes.

Une instanciation $\mathcal{I} = (a_1, a_2, ..., a_n)$ est un assignement de variables $\{v_1 = a_1, v_2 = a_2, \ldots, v_n = a_n\}$ où chaque variable v_i est assignée à une valeur a_i de son domaine D_i. Une contrainte $C_i \in C$ est satisfaite par $\mathcal{I}$ si la projection de $\mathcal{I}$ sur les variables impliquées dans C_i est un tuple de r_i. L'instanciation $\mathcal{I}$ est consistante si elle satisfait toutes les contraintes de C, donc $\mathcal{I}$ est une solution du CSP. Une instanciation d'un sous ensemble des variables CSP de V est appelée instanciation partielle. Chaque instanciation partielle $\mathcal{I}$ définit un noeud $n_{\mathcal{I}}$ dans l'arbre de recherche correspondant au CSP local $\mathcal{P}_{\mathcal{I}}$ résultant de $\mathcal{P}$ en considérant $\mathcal{I}$ et ses propagations induites. Une instanciation est totale si elle est définie sur toutes les variables. Etant donné un CSP, on cherche à décider sa consistance ou à énumérer toutes ces solutions (s'il en a). Plus de détail sur les CSPs peuvent être trouvés dans [26],[22] et [34].

Pour des soucis de simplicité, On a restreint notre étude au cas des CSPs binaires.

2.2 Permutations

Soit $\Omega = \{1, 2, \ldots, N\}$ pour un N donné, où chaque entier représente une variable CSP, une valeur, ou une paire variable/valeur. Une permutation de Ω est une application

bijective σ de Ω vers Ω. On dénote par $Perm(\Omega)$ l'ensemble de toutes les permutations de Ω et par $\circ$ la composition de permutation de $Perm(\Omega)$. La pair $(Perm(\Omega), \circ)$ forme le groupe de permutation de Ω. En effet, $\circ$ est close et associative, l'inverse d'une permutation est une permutation et la permutation identité est l'élément neutre. Une paire $(T, \circ)$ forme un sous-groupe de $(S, \circ)$ ssi T est un sous-ensemble de S et forme muni de l'opération $\circ$ un groupe.

L'orbite $\omega^{Perm(\Omega)}$ d'un élément ω de Ω sur lequel le groupe $Perm(\Omega)$ agit est $\omega^{Perm(\Omega)}$=$\{\omega^\sigma : \sigma \in Perm(\Omega)\}$.

Un ensemble de générateur du groupe $Perm(\Omega)$ est un sous-ensemble Gen de $Perm(\Omega)$ tel que chaque élément de $Perm(\Omega)$ peut être écrit comme composition des éléments de Gen. Nous écrivons $Perm(\Omega)$=$< Gen >$. Un élément de Gen est appelé générateur. L'orbite de $\omega \in \Omega$ peut être calculée en utilisant uniquement l'ensemble des générateurs Gen.

Un stabilisateur par point $Perm(\Omega)_\delta$ de $\delta \subset \Omega$ est le sous-groupe $Perm(\Omega)_\delta$
=$\{\sigma \in Perm(\Omega) : \forall \omega \in \delta, \omega^\sigma = \omega\}$, i.e. l'ensemble des éléments de $Perm(\Omega)$ qui fixent chaque point de δ.

Le sous-ensemble de permutations $Aut(\mathcal{M}_\mathcal{P}) \subset Perm(\Omega)$ préservant la microstructure $\mathcal{M}_\mathcal{P}$ d'un CSP $\mathcal{P}$ forment le groupe d'automorphisme de $\mathcal{M}_\mathcal{P}$ qui est connu pour être identique au groupe des *symétries de contrainte* du CSP $\mathcal{P}$ [9] que nous appelons *symétrie syntaxique* du CSP $\mathcal{P}$ dans [4].

3 Symétrie Sémantique

Comme nous nous intéressons aux deux problèmes dans les CSPs : le problème de la recherche d'une solution (consistance) du CSP et le problème de la recherche de toutes les solutions du CSP, nous définissons deux niveaux de symétrie sémantique.

Définition 1 (Symétrie sémantique pour la consistance) *Deux valeurs b_i et c_j du domaine des variables CSP $v_i \in V$ respectivement $v_j \in V$ sont symétriques pour la consistance ssi les assertions suivantes sont équivalentes :*

1. *Il existe une solution du CSP qui assigne la valeur b_i à la variable v_i ;*
2. *Il existe une solution du CSP qui assigne la valeur c_j à la variable v_j.*

En d'autres mots, l'affectation $v_i = b_i$ participe à une solution du CSP si et seulement si l'affectation $v_j = c_j$ participe dans une solution aussi ; autrement aucune des deux ne participe.

En plus d'être symétriques pour la consistance (définition 1), les valeurs d'un domaine peuvent être symétriques pour toutes les solutions. Donc, si $sol(\mathcal{P})$ dénote l'ensemble des solutions du CSP $\mathcal{P}$, alors nous définissons un second niveau de symétrie sémantique comme suite :

Définition 2 (Symétrie sémantique pour toutes les solutions) *Les deux valeurs b_i et c_j des domaines des variables CSP $v_i \in V$ respectivement $v_j \in V$ sont symétriques pour $sol(\mathcal{P})$ si et seulement si chaque solution du CSP assignant la valeur b_i à v_i peut être transformée en une solution assignant la valeur c_j à v_j et vice-versa.*

Ceci veut dire que l'ensemble des solutions dans lesquelles l'affectation $v_i = b_i$ participe est isomorphe à celles dans lesquelles $v_j = c_j$ participe. Ce sont des solutions symétriques. Donc si nous cherchons toutes les solutions du CSP $\mathcal{P}$, les valeurs symétriques pour $sol(\mathcal{P})$ nous permet de trouver des familles de solutions symétriques.

Remarque 1
1. *Si les variables v_i et v_j désignent la même variable, alors les deux définitions précédentes concernent les symétries de valeurs d'un même domaine.*
2. *La symétrie pour toutes les solutions implique la symétrie pour la consistance.*

Identifier la symétrie sémantique comme définie dans la section 3 est clairement très coûteux, car cela demande de résoudre le problème. Nous étudions dans la section suivante la notion de la symétrie syntaxique qui est plus facilement calculable et qui est une condition suffisante pour la symétrie sémantique.

4 Symétrie syntaxique

Dans [4], nous avons étudié la notion de symétrie syntaxique pour les valeurs d'un même domaine, ici cette notion est étendue aux valeurs des différents domaines appelées *La symétrie inter-domaines.*

Définition 3 *Une symétrie syntaxique inter-domaines d'un CSP $\mathcal{P} = (V, D, C, R)$ est une application $\sigma : \cup_{i\in[1,n]} D_i \longrightarrow \cup_{i\in[1,n]} D_i$, qui associe à $d_i \in D_i$, la valeur $\sigma(d_i) = d_j \in D_j$, où D_i et D_j sont des domaines des variables CSP v_i et v_j, et qui garde le CSP $\mathcal{P}$ invariant.*

Une symétrie entre valeurs de différentes variables peut induire une symmetry entre les variables elles-mêmes. Ceci peut arriver quand chaque valeur de la variable v_i est couplée avec une valeur de v_j en appliquant la permutation σ, alors v_i est couplée avec v_j dans l'ensemble des contraintes, et donc une symétrie σ_V sur les variables est induite par σ.

Pour garder le CSP invariant, l'ensemble des contraintes doit rester le même après la permutation. La permutation σ doit vérifier deux conditions : $\sigma_C(C) = C$ et $\sigma_R(R) = R$

où σ_C resp. σ_R sont les permutations induites par σ sur les ensembles C resp. R. En effet, $\forall r_{ik} \in R$, si $< d_i, d_k > \in$ tuples(r_{ik}) alors $< \sigma(d_i), \sigma(d_k) > \in$ tuples($\sigma_R(r_{ik})$).

Remarque 2 *Il faut noter que dans le cas de symétrie entre valeurs d'un même domaine, σ_R est l'application identité, i.e $\sigma(r_{ij}) = r_{ij}$, $\forall r_{ij} \in R$.*

Les symétries syntaxiques entre valeurs de différents domaines d'un CSP $\mathcal{P}$ sont des automorphismes de la microstructure $\mathcal{M}_\mathcal{P}$ qui sont *des symétries de contraintes* [9]. L'ensemble des *symétries de contraintes* est équivalent au groupe d'automorphisme $Aut(\mathcal{M}_\mathcal{P})$.

Theorème 1 *Si deux valeurs $b_i \in D_i$ et $c_j \in D_j$ des variables CSPs $v_i \in V$ et $v_j \in V$ sont syntaxiquement symétriques de valeurs syntaxiques, alors b_i et c_j sont sémantiquement symétriques pour toutes les solutions du CSP.*

Preuve 1 *Nous avons à prouver que chaque solution dans laquelle l'affectation $v_i = b_i$ apparaît peut être transformée en une autre solution dans laquelle l'affectation $v_j = c_j$ apparaît (Définition 2) et vice-versa. Si les valeurs b_i et c_j sont syntaxiquement symétriques (ce qui est vrai par hypothèses), alors il existe une symétrie syntaxique σ du CSP $\mathcal{P}$ telle que $\sigma(b_i) = c_j$. Maintenant, supposons que l'affectation $v_i = b_i$ apparaît dans une solution I du CSP $\mathcal{P}$, alors $\sigma(I)$ est une solution dans laquelle l'affectation $v_j = c_j$ apparaît, puisque la symétrie σ préserve les solutions. En d'autres mots, la solution I est transformée en $\sigma(I)$ en utilisant la symétrie σ. Nous prouvons l'inverse en considérant la symétrie inverse σ^{-1}. Donc b_i et c_j sont des valeurs sémantiquement symétriques pour toutes les solutions(CQFD).*

Quand une instanciation partielle ne génère pas de conflit, alors les symétries syntaxiques locales peuvent être utilisées pour éviter de générer des solutions localement symétriques. Ceci peut être fait en considérant uniquement une valeur dans chaque classe des valeurs symétriques dans le domaine de la variable courante durant la résolution.

Dans ce qui suit nous montrons comment les symétrie syntaxiques locales sont détectées et éliminées dynamiquement, et comment une méthode de résolution peut exploiter efficacement ces symétries.

5 Détection et exploitation des symétries syntaxiques locales

5.1 Détection

Dans de nombreuses méthodes exploitant les symétries globales, les symétries sont supposées être connues à l'avance et données donc comme entrée supplémentaire avec les autres données du problème. Ces méthodes utilisent pour la plupart l'outil GAP (Group Automorphism package) pour la recherche des symétries. Mais , la détection des symétries n'est pas totalement automatisée avec l'outil GAP. En effet, c'est à l'utilisateur de poster les symétries primaires à la main qui seront utilisées par GAP pour générer le groupe de symétrie $correcte$ du problème. Cette approche ne peut pas être utilisée dans le cas de symétries locales, puisque les symétries locales doivent être détectées dynamiquement à chaque noeud de l'arbre de recherche.

La détection dynamique des symétries a été étudié pour les CSPs, une méthode de détection des symétries locales a été proposée dans [4]. Cette méthode se divise en deux étapes élémentaires : la première consiste à partitionner chaque domaine par rapport à la condition nécessaire de symétrie[5] sous forme de classes primaires de valeurs candidates. La seconde étape consiste à calculer une permutation σ sur les classes primaires résultantes de la première étape qui garde le CSP invariant.

Comme alternative à cette dernière méthode, nous avons adapté Saucy [1] pour la détection des symétries syntaxiques locales entre valeurs d'un même domaine et nous montrons comment exploiter les symétries trouvées au cours de la résolution. Saucy est un outil qui sert à calculer le groupe d'automorphisme d'un graphe. D'autres outils comme Nauty [23] ou le plus récent AUTOM [28] ou celui qui est décrit dans [24] peut être adapté pour la recherche des symétries locales. Il est montré dans [28] que AUTOM est le plus performant or le code source de AUTOM n'est pas libre. Pour nos expérimentations nous avons choisit d'utiliser Saucy. Comme le groupe de symétries syntaxiques d'un CSP $\mathcal{P}$ est identique au groupe d'automorphismes de sa microstructure $\mathcal{M}_\mathcal{P}$, nous pouvons utiliser Saucy sur $\mathcal{M}_\mathcal{P}$ pour détecter le groupe de symétries syntaxiques de $\mathcal{P}$.

Saucy retourne un ensemble de générateurs Gen du groupe des symétries duquel on peut déduire chaque symétrie. Saucy offre la possibilité de colorer la microstructure tel qu'un sommet n'est autorisé à être permuter qu'avec un autre sommet s'ils ont la même couleur. Ceci restreint les permutations à celles qui se font sur des sommets de même couleur. Nous utilisons cette possibilité de coloration pour guider la recherche de symétries et pour détecter les symétries de valeurs locales. Le code source de Saucy peut être téléchargé à l'adresse http ://vlsicad.eecs.umich.edu/BK/SAUCY/.

5.1.1 Détection des symétries grâce à Saucy

Soit le CSP $\mathcal{P}$, et une instanciation partielle $\mathcal{I}$ de $\mathcal{P}$, définissant un état dans la résolution correspondant au noeud courant $n_\mathcal{I}$. L'idée est de maintenir dynamiquement la microstructure $\mathcal{M}_{\mathcal{P}_\mathcal{I}}$ du CSP $\mathcal{P}_\mathcal{I}$ correspondant au sous pro-

[5] Deux valeurs b_i et c_i sont candidates pour la symétrie si elles ont le même nombre d'occurrences dans chaque relation $r \in R$ du CSP.

blème local défini à chaque noeud courant $n_{\mathcal{I}}$, puis colorer la microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ et calculer son groupe d'automorphismes $Aut(\mathcal{M}_{\mathcal{P}_{\mathcal{I}}})$. Le CSP $\mathcal{P}_{\mathcal{I}}$ peut être vu comme nouveau problème correspondant à la partie non résolue de $\mathcal{P}$. Calculer tous les automorphismes de la microstructure dynamique à chaque noeud de l'arbre de recherche peut être coûteux. Pour y remédier, deux stratégies de coloration de la microstructure sont considérées :

1. **La stratégie multi couleurs** : Un premier compromis est de limiter les permutations aux valeurs d'un même domaine. Pour faire cela, une couleur est associée à chaque variable. Chaque sommet de la microstructure appartenant à la même variable est coloré avec une même couleur. Maintenant en appliquant Saucy sur cette microstructure colorée nous pouvons récupérer l'ensemble des générateurs Gen du sous-groupe des symétries existant entre les valeurs d'un même domaine du CSP.
2. **La stratégie deux coloration** : Un second compromis est d'associer à la variable courante v_i une couleur et aux autres variables une même et une seule couleur. En effet, nous colorons la microstructure dynamique $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ avec deux couleurs. Tous les sommets de la microstructure appartenant à la variable courante v_i ont la première couleur et tous les autres sommets ont la deuxième couleur. Finalement, nous appliquons Saucy pour calculer les générateurs du sous-groupe d'automorphismes correspondant à cette coloration. Ceci retourne les générateurs Gen du sous-groupe des symétries permettant les permutations inter-domaines sur les autres variables qui différent de v_i, mais les valeurs de v_i sont permutées entres elles.

Remarque 3 *Le groupe total des symétrie est atteint quand la microstructure $\mathcal{M}_{\mathcal{P}_{\mathcal{I}}}$ n'est colorée qu'avec une seule couleur .*

5.1.2 Elimination des Symétries

Durant la recherche, le domaine D_i de la variable courante v_i est partitionnée en des sous-ensembles de valeurs localement symétriques. Pour éviter de générer les solutions localement symétriques, nous considérons une valeur de chaque sous-ensemble de valeurs symétriques dans D_i.

Si nous sommes intéressés uniquement par la consistance du CSP alors nous arrêtons la recherche dès qu'une solution est trouvée.

5.2 Exploitation

Nous pouvons maintenant montrer comment ces symétries peuvent être utilisées pour augmenter l'efficacité des méthodes de résolution. Nous choisissons comme base de notre implémentation la méthode *Forward Checking* dont nous voulons améliorer l'efficacité.

Si $\mathcal{I}$ est une instanciation partielle inconsistante dans laquelle l'affectation de la variable courante $v_i = d_i$ ne participe à aucune solution du CSP $\mathcal{P}$, alors par application du Théorème 1, toutes les valeurs qui sont symétriques à d_i dans $\mathcal{P}_{\mathcal{I}}$ ne participent à aucune solution aussi. Donc, nous élaguons l'espace qui correspond à leurs affectations.

Nous pouvons couper n-1 branches dans l'arbre de recherche s'il y a n valeurs symétriques et qu'une d'entres elles a été testée et montrée comme ne participant à aucune solution du CSP. Si $SymClass(d)$ dénote la classe des valeurs qui sont symétriques à d (i.e. l'orbite de d), alors lors de la résolution, nous ne considérons que la valeur d, puisque les autres valeurs de $SymClass(d)$ sont symétriques à cette valeurs. L'élimination des symétries est intégrée dans un Forward Checking et la procédure résultante FC-sym est donnée dans la Figure 1.

Bien sûr, d'autre méthodes comme MAC (Maintaining Arc Consistency) peuvent être adaptées pour exploiter la symétrie.

Procedure FC-sym($D, \mathcal{I}, k$);{$\mathcal{I} = [d_1, d_2, \ldots, d_k]$}
begin
  if $k = n$ **then** $[d_1, d_2, \ldots, d_k]$ is a solution, return($\mathcal{I}$)
  else
  begin
    for each $v_i \in V$, such that $C_{ik} \in C, v_i \in$future(v_k) **do**
      for each value $d_i \in D_i$ **do**
        if $(d_i, d_k) \notin r_{ik}$ **then**
          delete d_i from D_i ;
    if $\forall v_i \in future(v_k), D_i \neq \emptyset$ **then**
    begin
      v_{k+1}=next-variable(v_k)
      repeat
        take $d_{k+1} \in D_{k+1}$
        $D_{k+1} = D_{k+1} - d_{k+1}$
        $\mathcal{I} = [d_1, d_2, \ldots, d_k, d_{k+1}]$;
        $\mathcal{J}$=FC-sym($D, \mathcal{I}, k+1$) ;
        $\mathcal{I}$=$\mathcal{I}$-d_{k+1} ;
        Gen=Saucy($\mathcal{P}_{\mathcal{I}}$) ;
        SymClass(d_{k+1})=orbit(d_{k+1},Gen) ;
        D_{k+1}=D_{k+1}-SymClass(d_{k+1})
      until $D_{k+1} = \emptyset$
    end
  end
end

FIG. 1 – Forward Checking avec détection et élimination de la symétrie

La fonction $orbit(d_{k+1}, Gen)$ est élémentaire, elle calcule l'orbite de la valeur d_{k+1} à partir de l'ensemble des générateurs Gen retourné par Saucy.

6 Expérimentations

Nous avons choisit pour nos expérimentations des problèmes classiques. Nous pensons que l'élimination des symétries pourrait être meilleur sur des problèmes réels. Ici, nous avons testé et comparé quatre méthodes :

1. **No-sym :** recherche sans élimination de symétries ;
2. **Global-sym :** recherche avec élimination des symétries globales restreinte aux valeurs d'un même domaine. Les mêmes symétries que celles considérées dans la méthode GE-tree, avec une légère différence (nous considérons que les symétries globales entre valeurs d'un même domaine) ;
3. **Local-sym1 :** recherche avec élimination des symétries de valeurs locales. Cette méthode implémente la stratégie multi couleurs décrite dans la Section 5.1.
4. **Local-sym2 :** recherche avec élimination des symétries locales inter-domaines. Cette méthode implémente la stratégie de deux coloration décrite dans Section 5.1.

sur les problèmes suivant : problèmes aléatoires de coloration de graphes, des instances de colorations de graphes de Dimacs et les problèmes des n-Reines. Une implémentation de la stratégie $Local - sym1$ a été implémenté avec succès sous GECODE dans [37] afin de casser les symétries locales pour le problème de l'appariement de graphes. Le point commun entre les quatre méthodes est que la méthode support est le Forward Checking. Les indicateurs de complexité sont le nombre de noeuds de l'arbre de recherche et le temps CPU. Le temps nécessaire pour le calcul des symétries est ajouté au temps total. Le code source a été écrit en C et les tests ont été effectués sur une machine muni d'un Pentium 4, 2.8 GHZ et de 1 Go de RAM.

6.1 Problèmes aléatoires de coloration de graphes

Les problèmes sont générés aléatoirement en fixant les paramètres suivants : (1) n : le nombre de sommets (variables), (2) $Colors$: le nombre de couleurs (valeurs) et (3) d : la densité qui est une valeur entre 0 et 1, exprimant le ratio : le nombre de contraintes (le nombre des arêtes dans le graphe de contraintes) sur le nombre maximal de contraintes. Pour chaque test correspondant à n, $Colors$ et d fixés, un échantillon de 100 problèmes sont générés aléatoirement. Les mesures (temp CPU, nombre de noeuds) sont données en moyenne.

La Figure 2 montre les performances des quatre méthodes en nombre de noeuds moyen, respectivement, en temps CPU moyen (en secondes) sur les problèmes aléatoires de coloration de graphes où le nombre de variables est fixé à $n = 15$ et la densité à $d = 0.9$. Les courbes du dessus sont tracées avec une échelle logarithmique, elles

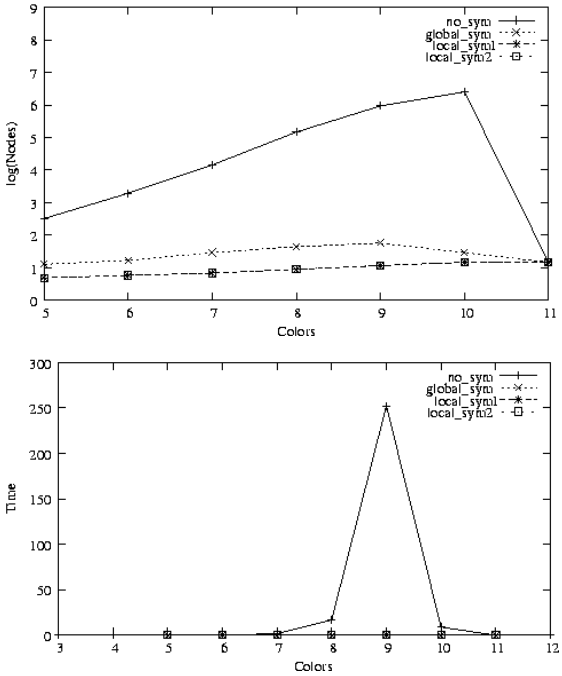

FIG. 2 – Courbes de noeuds et de temps CPU où $n = 15$ et $d = 0.9$

représentent les performances en nombre de noeuds en fonction du nombre de couleurs. Celles en dessous ont été tracées avec l'échelle usuelle et expriment les performances en temps CPU moyen en fonction du nombre de couleurs. On voit bien que les méthodes qui exploitent la symétrie sont très largement plus performantes que la méthode (No-sym) en nombre de noeuds ainsi qu'en temps CPU. Nous pouvons voir aussi que les deux méthodes qui exploitent les symétries locales (Local-sym1 et Local-sym2) réduisent plus l'espace de recherche que celle qui n'élimine que les symétries globales (Global-sym). En effet, les symétries locales sont plus fréquentes durant la recherche que les symétries globales (stabilisant l'instanciation partielle). Les deux méthodes Local-sym1 et Local-sym2 ont globalement les mêmes performances, leurs courbes se confondent. Nous pouvons distinguer la courbe correspondante au temps CPU de la méthode No-sym au voisinage du pic de difficulté où les instances sont les plus difficiles. Toutes les méthodes utilisant les symétries ont résolues ces problèmes en moins de 0.1 secondes, et donc leur courbes des temps CPU se confondent avec l'axe des abscisses.

Comme la Figure 2 ne permet pas une comparaison entre les temps CPU des méthodes exploitant la symétrie, nous avons reporté dans la Figure 3 les résultats des méthodes : Global-sym, Local-sym1 et Local-sym2, sur des problèmes où le nombre de variables est fixé à $n = 35$, pour la même densité ($d = 0.9$) comme dans la Figure 2.

Nous pouvons déduire à partir des courbes (celles du

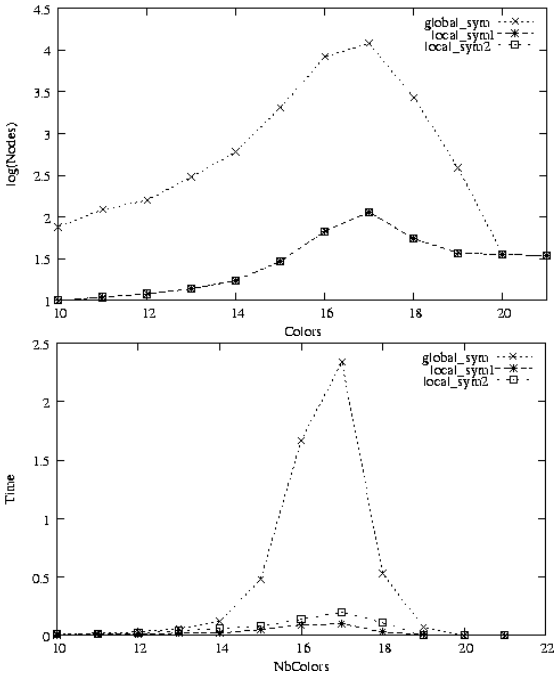

FIG. 3 – Courbes de noeuds et de temps CPU des trois méthodes exploitant la symétrie où $n = 35$ et $d = 0.9$

dessus sont en échelle logarithmique) des méthodes Local-sym1 et Local-sym2 qu'elles détectent et éliminent plus de symétries que la méthode Global-sym. La raison est, comme dite précédemment, que les symétries locales détectées en un noeud durant la recherche par Local-sym1 et Local-sym2 inclus les symétries globales stabilisant l'instanciation partielle correspondant à ce noeud et qui sont exploitées par Global-sym. Les courbes du nombre de noeuds moyen des deux méthodes Local-sym1 et Local-sym2 sont comparables. A partir des courbes des temps CPU (les courbes du dessous), nous pouvons voir que Local-sym1 et Local-sym2 sont plus rapides que Global-sym. Au voisinage du pic de difficultés, Local-sym2 semble être 12 fois plus rapide que Global-sym et Local-sym1 près de 24 fois de plus que Global-sym. Ce qui nous permet de dire que l'élimination des symétries locales semble être profitable pour la résolution des problèmes de coloration de graphes dans la région critique correspondant aux problèmes les plus dures et que les performances des méthodes exploitant la symétrie locale sont plus élevées que la performance de la méthode qui n'exploitent que la symétrie globale sur ces problèmes. Nous pouvons voir aussi que sur le temps CPU que Local-sym1 est plus performante que Local-sym2, donc le bon compromis semble être celui décrit dans la stratégie multi couleurs correspondant aux symétries de valeurs d'un même domaine. Ces remarques seront confirmées par les expérimentations sur les instances de Dimacs dans la Section suivante.

6.2 Instances de coloration de graphes de Dimacs

		No-sym		Global-sym	
Instance	k	*Nodes*	*Time*	*Nodes*	*Time*
myciel4	5	30,976	0.16	2,764	0.03
myciel5	6	-	-	8,040,259	59.84
anna	11	-	-	3,403	0.59
david	11	-	-	3,896	0.23
queen7_7	7	2,452	0.01	513	0.02
queen8_8	9	-	-	10,629,131	262.54
school1	14	-	-	-	-
school1_nsh	14	-	-	-	-
2-Insertion_3	4	832,150	1.02	277,408	0.73
2-FullIns_3	5	2,294,396	7.63	193,347	1.14
mugg88_1	4	-	-	-	-
mugg88_25	4	-	-	-	-
mugg100_1	4	-	-	-	-
mugg100_25	4	-	-	-	-
zeroin.i.1	49	-	-	468	0.1
zeroin.i.2	30	-	-	-	-
zeroin.i.3	30	-	-	-	-
mulsol.i.2	31	-	-	-	-
mulsol.i.3	31	-	-	-	-
le450_5a	5	178,753	13.88	170,123	11.1
le450_5b	5	1,349	0.11	1,110	0.09
le450_5c	5	1,984	0.15	1,984	0.17
le450_5d	5	5,795	0.54	4,563	0.34
DSJC125.1	5	55,358	0.85	43,773	1.34

		Local-sym1		Local-sym2	
Instance	k	*Nodes*	*Time*	*Nodes*	*Time*
myciel4	5	1,260	0.01	1,260	0.04
myciel5	6	2,413,556	22.21	2,406,945	25.36
anna	11	168	0.05	168	0.08
david	11	124	0.03	124	0.03
queen7_7	7	502	0.01	502	0.0
queen8_8	9	1,399,436	29.7	1,396,774	30.16
school1	14	76,192	17.28	75,985	17.85
school1_nsh	14	1,487,287	257.57	1,486,523	270.4
2-Insertion_3	4	135,953	0.48	115,737	0.52
2-FullIns_3	5	49,202	0.59	48,076	0.65
mugg88_1	4	2,882,284	53.91	2,882,284	93.55
mugg88_25	4	881,784	6.74	881,784	9.4
mugg100_1	4	3,325,453	24.85	3,325,453	40.15
mugg100_25	4	2,727,178	17.3	2,727,178	30.92
zeroin.i.1	49	268	7.0	268	35.49
zeroin.i.2	30	262	0.75	262	3,675
zeroin.i.3	30	262	0.76	262	3,675
mulsol.i.2	31	237	0.85	237	10.14
mulsol.i.3	31	237	0.9	237	10.14
le450_5a	5	167,787	32.0	167,703	32.23
le450_5b	5	927	0.11	927	0.19
le450_5c	5	1,983	0.31	1,975	0.34
le450_5d	5	3,452	0.62	3,452	0.68
DSJC125.1	5	40,809	1.44	40,809	1.48

TAB. 1 – Résultats obtenue sur des instances de Dimacs

Ici, nous avons testé et comparé les quatre méthodes sur des problèmes de coloration de graphes tirés du challenge de Dimacs (http ://mat.gsia.cmu.edu/COLOR04/).

La Table 1 montre les résultats obtenus des différentes méthodes sur des problèmes de Dimacs. On donne l'instance, le nombre chromatique trouvé (k), le nombre de noeuds de l'arbre de recherche et le temps CPU pour chaque méthode. Nous cherchons pour chaque instance le nombre minimal k de couleurs nécessaire pour colorer les sommets du graphe correspondant (appelé le nombre chromatique). La recherche de ce nombre consiste à prouver la consistance du problème avec k couleurs (existence d'une k-coloration du graphe), et à prouver l'inconsistance avec $k - 1$ couleurs (n'est pas colorable). Le symbole "-" signifie que la méthode correspondante n'a pas résolue l'instance après 1 heure de calcul. Le nombre de noeuds affiché est la somme du nombre de noeuds nécessaire pour trouver une k-coloration et du nombre de noeuds nécessaire pour prouver qu'il n'y a pas de $k - 1$-coloration (idem pour le temps CPU). La Table 1 montre que les méthodes No-sym et Global-sym ne sont pas arrivées à résoudre plusieurs instances. Mais, Global-sym est meilleure que No-sym en

nombre de noeuds et en temps CPU. Nous pouvons voir que Local-sym1 et Local-sym2 sont meilleures que Global-sym en nombre de noeuds et en temps CPU. En effet, l'élimination des symétries locales est plus profitable que l'élimination des symétries globales seules. Nous pouvons voir aussi que Local-sym1 élimine les mêmes symétries que celles éliminées par Local-sym2, mais Local-sym1 est plus rapide. Ceci confirme que la stratégie implémentée dans Local-sym1 (la stratégie multi couleurs) donne le meilleur compromis sur ces problèmes.

6.3 Les problèmes des n-Reines

Ce problème est un problème classique qui consiste à trouver une façon de placer les n reines sur un échiquier $n \times n$ telles qu'aucune reine ne soit prise par une autre. Trouver *toutes* les solutions de ce problème reste un challenge. Nous avons comparé les quatre méthodes sur certaines instances de ce problème.

	No-sym			Global-sym		
n	$Sols$	$Nodes$	$Time$	$Sols$	$Nodes$	$Times$
8	92	1,360	0.0	46	680	0.01
9	352	5,399	0.0	179	2,800	0.0
10	724	19,744	0.03	362	9,872	0.03
11	2,680	85,939	0.1	1,382	43,958	0.07
12	14,200	416,828	0.28	7,100	208,414	0.25
13	73,712	2,154,845	2.69	37,361	1,093,606	1.99
14	365,596	11,799,746	46.95	51,726	5,899,873	20.65
	Local-sym1			Local-sym2		
n	$Sols$	$Nodes$	$Time$	$Sols$	$Nodes$	$Times$
8	45	664	0.01	45	662	0.01
9	172	2,645	0.02	168	2,625	0.05
10	355	9,656	0.07	353	9,640	0.08
11	1,309	42,154	0.25	1,305	42,078	0.31
12	6,883	204,901	2.05	6,839	203,611	2.19
13	35,525	1,055,366	11.44	35,312	1,053,053	11.58
14	44,334	5,777,244	69.6	43,257	5,765,594	75.6

TAB. 2 – Resultats obtenus sur le problème des n-Reines

La Table 2 regroupe les résultats obtenus. Pour chaque méthode nous donnons le nombre de solutions calculées ($Sols$), le nombre de noeuds, et le temps CPU en seconde. A noter que pour les méthodes exploitant la symétrie, le nombre de solutions ($Sols$) est le nombre de solutions non symétriques trouvées en fonction de la stratégie de symétrie appliquée. Le nombre de solutions de No-sym est le nombre total de solutions du problème donné. Nous pouvons voir que Local-sym1 et Local-sym2 compactent plus l'ensemble des solutions que Global-sym. Ceci signifie que Local-sym1 et Local-sym2 compactent des solution localement symétriques en plus des solutions globalement symétriques compactées par Global-sym. Nous pouvons aussi voir que Local-sym2 détecte des solutions localement symétriques qui ne sont pas considéré ou détectées par Local-sym1. Ceci est du à certaines symétries inter-domaines locale qui sont détectés dans Local-sym2, mais pas dans Local-sym1. Maintenant, si on compare globalement les méthodes en nombre de solutions, en nombre de noeuds et temps CPU, la méthode Global-sym semble être la meilleure en moyenne. En plus, la symétrie globale est suffisante pour résoudre efficacement ces problèmes. Nous pensons que l'exploitation des symétries locales de variables donnerait de meilleurs résultats sur les n-Reines.

7 Discussion et conclusions

Ici, nous avons étendu le principe de détection et d'élimination des symétries locales. En effet, les symétries de chaque sous-CSP correspondant à une instanciation partielle du CSP initial sont détectées et éliminées. Nous avons adapté Saucy pour le calcul de ces symétries locales en maintenant dynamiquement la microstructure du sous-CSP définit en chaque noeud de l'arbre de recherche. Contrairement aux méthodes utilisant GAP, ici, la détection des symétries locales est totalement automatisée. Saucy est appelé avec la microstructure du sous-CSP local comme paramètre, et retourne l'ensemble des générateurs du groupe d'automorphismes qui est montré être équivalent au groupe de symétries locales du sous-CSP considéré. Détecter et exploiter tout le groupe des symétries locales des différents noeuds durant la recherche peut être coûteux. Pour y remédier, nous avons introduit deux stratégies de coloration de la microstructure afin de guider la détection et de réduire l'espace des permutations. La première stratégie (multi couleurs) qui restreint la recherche des symétries aux valeurs d'un même domaine, et la deuxième stratégie (deux coloration) qui restreint la recherche a une partie des symétries inter-domaines. Les expérimentations ont montrées que l'élimination des symétries locales est profitable pour la résolution des CSPs et que les résultats obtenus sont meilleurs que ceux de la méthode exploitant que les symétries globales sur la plupart des problèmes considérés.

Un point intéressant à voir dans le futur est d'étendre la détection des symétries locales aux symétries locales de variables. Ce serait important d'étudier la combinaison des symétries locales de valeurs et les symétrie locales de variables.

Finalement, nous sommes intéressés d'exporter l'élimination des symétries locales dans d'autres domaines de recherche tels que la biologie et la recherche opérationnelle afin de s'attaquer à des problèmes réels.

Références

[1] F. A. Aloul, A.Ramani, I. L. Markov, and K. A. Sakallak. Solving difficult sat instances in the presence of symmetry. *In IEEE Transaction on CAD, vol. 22(9)*, pages 1117–1137, 2003.

[2] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallak. Symmetry breaking for pseudo-boolean satisfiabilty. *In ASPDAC'04*, pages 884–887, 2004.

[3] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In *Principle and Practice of Constraint Programming - CP'99*, 1999.

[4] B. Benhamou. Study of symmetry in constraint satisfaction problems. *In Proceedings of the 2nd International workshop on Principles and Practice of Constraint Programming - PPCP'94*, 1994.

[5] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *Eleventh International Conference on Automated Deduction, Saratoga Springs,NY, USA*, 1992.

[6] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning (JAR),*, 12 :89–102, 1994.

[7] B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language. *in proceedings of STACS'94, Caen France*, pages 71–82, 1994.

[8] Belaïd Benhamou and Mohamed Réda Saïdi. Eliminating local symmetries in csp. In *The International Symmetry Conference (ISC 2007)*, Edinburgh, SCOTLAND, january 2007. (POSTER).

[9] D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. *In, proceedings of CP*, pages 17–31, 2005.

[10] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *KR'96 : Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.

[11] Olivier Dubois and Gilles Dequen. The non-existence of (3,1,2)-conjugate orthogonal idempotent Latin square of order 10. In *7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *LNCS*, pages 108–121. Springer Verlag, 2001.

[12] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 93–108. Springer Verlag, 2001.

[13] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 77–82. Springer Verlag, 2001.

[14] E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233, 1991.

[15] I. P. Gent, W. Hervey, T. Kesley, and S. Linton. Generic sbdd using computational group theory. In *Proceedings CP'2003*, 2003.

[16] I. P. Gent, T. Kelsey, S. A. Linton, I. McDonald, I. Migeul, and B.Smith. Conditional symmetry breaking. In *Principle and Practice of Constraint Programming - CP 2005*, pages 256–270, 2005.

[17] I. P. Gent and B. M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings ECAI'2000*, 2000.

[18] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints : Symmetry breaking during search. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 415–430. Springer Verlag, 2002.

[19] P. Jegou. Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. In *In Proceedings AAAI'93*, 1993.

[20] B. Krishnamurty. Short proofs for tricky formulas. *Acta informatica*, (22) :253–275, 1985.

[21] J. Slaney M. Fujita and F. Bennett. Automatic generation of some results in finite algebra. *In proceedings of the 13th Internationnal Joint Conference on Artificial Intelligence, Chambery, France*, pages 52–57, 1993.

[22] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence 8*, pages 99–118, 1977.

[23] B McKay. Practical graph isomorphism. In *Congr. Numer. 30*, pages 45–87, 1981.

[24] C. Mears, M. Garcia de la Banda, and M. Wallace. On implementing symmetry detection. In *The CP 2006 Workshop on Symmetry and Constraint Satisfaction Problems (SymCon'06)*, pages 1–8, Cité des Congrès - Nantes, France, septembre 2006.

[25] Pedro Meseguer and Carme Torras. Solving strategies for highly symmetric csps. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 400–405. Morgan Kaufmann, 1999.

[26] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Science 7*, pages 95–132, 1974.

[27] J. F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *In J. Kamorowski and Z. W. Ras,editors, Proceedings of ISMIS'93, LNAI 689*, 1993.

[28] J F Puget. Automatic detection of variable and value symmetries. In LNCS Springer, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 474–488, Sitges, Spain, october 2005.

[29] J.F. Puget. Symmetry breaking revisited. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 446–461. Springer Verlag, 2002.

[30] J.F. Puget. Breaking all value symmetries in surjection problems. *In, proceedings of CP*, pages 490–504, 2005.

[31] J.F. Puget. Breaking symmetries in all diffrent problems. *In, proceedings of IJCAI*, pages 272–277, 2005.

[32] J.F. Puget. Dynamic lex constraints. *In, proceedings of CP'06*, pages 453–467, 2006.

[33] C. M. Roney-Dougal, I. P. Gent, T. Kelsey, and S. A. Linton. Tractable symmetry breaking using restricted search trees. *In, proceedings of ECAI'04*, pages 211–215, 2004.

[34] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, ISBN 0-12-701610-4, 1993.

[35] T. Walsh. General symmetry breaking constraints. *In, proceedings of CP'06*, pages 650–664, 2006.

[36] S. Zampelli, Y. Deville, and P. Dupont. Symmetry breaking in subgraph pattern matching. In *SymCon'06*, pages 35–42, 2006.

[37] Stéphane Zampelli, Yves Deville, Mohamed Réda Saïdi, Belaïd Benhamou, and Pierre Dupont. Breaking local symmetries in subgraph pattern matching. In *The International Symmetry Conference (ISC 2007)*, Edinburgh, SCOTLAND, january 2007.